A Game of Words

Table of Contents

Load & Import Data

> `Sumerians invented the wheel. I'd much rather just use it.`

In [3]:
```
import io
import numpy as np
import os
import string
from tqdm import tqdm
import re
import tensorflow as tf
from tensorflow.keras import Model, Sequential
from tensorflow.keras.layers import Activation, Dense, Dot, Embedding, Flatten, Reshape
from tensorflow.keras.layers.experimental.preprocessing import TextVectorization
import pathlib
from sklearn.manifold import TSNE
from tensorboard.plugins import projector

import matplotlib.pyplot as plt
%matplotlib inline
```

In [4]:
```
base_dir = pathlib.Path(os.getcwd())
data_dir = base_dir / 'Data'
train_file = [file for file in pathlib.Path.glob(data_dir, pattern = '*train.tokens')][0]
test_file = [file for file in pathlib.Path.glob(data_dir, pattern = '*test.tokens')][0]
valid_file = [file for file in pathlib.Path.glob(data_dir, pattern = '*valid.tokens')][0]
```

In [14]:
```
with open(train_file, encoding = 'UTF-8') as f:
    lines1 = f.read().splitlines()

with open(test_file, encoding = 'UTF-8') as f:
    lines2 = f.read().splitlines()

with open(test_file, encoding = 'UTF-8') as f:
    lines3 = f.read().splitlines()

lines = list(np.concatenate((np.concatenate((lines1, lines2), axis = 0), lines3), axis = 0))
```

In [19]:
```
text_ds = tf.data.TextLineDataset([train_file, test_file, valid_file]).filter(lambda x:tf.cast(tf.strings.lengt
text_ds = text_ds_1.filter(lambda x: tf.cast(x != b' ', bool))
```

In [20]:
```
sentences = list(text_ds.as_numpy_iterator())
print(len(sentences))
```

```
29119
```

In [24]:
```
for stc in sentences[:5]:
    print(stc)
```

```
b' = Valkyria Chronicles III = '
b' Senj\xc5\x8d no Valkyria 3 : <unk> Chronicles ( Japanese : \xe6\x88\xa6\xe5\xa0\xb4\xe3\x81\xae\xe3\x83\xb4
\xe3\x82\xa1\xe3\x83\xab\xe3\x82\xad\xe3\x83\xa5\xe3\x83\xaa\xe3\x82\xa23 , lit . Valkyria of the Battlefield 3
) , commonly referred to as Valkyria Chronicles III outside Japan , is a tactical role @-@ playing video game d
eveloped by Sega and Media.Vision for the PlayStation Portable . Released in January 2011 in Japan , it is the
third game in the Valkyria series . <unk> the same fusion of tactical and real @-@ time gameplay as its predece
ssors , the story runs parallel to the first game and follows the " Nameless " , a penal military unit serving
the nation of Gallia during the Second Europan War who perform secret black operations and are pitted against t
he Imperial unit " <unk> Raven " . '
b" The game began development in 2010 , carrying over a large portion of the work done on Valkyria Chronicles I
I . While it retained the standard features of the series , it also underwent multiple adjustments , such as ma
king the game more <unk> for series newcomers . Character designer <unk> Honjou and composer Hitoshi Sakimoto b
oth returned from previous entries , along with Valkyria Chronicles II director Takeshi Ozawa . A large team of
writers handled the script . The game 's opening theme was sung by May 'n . "
b" It met with positive sales in Japan , and was praised by both Japanese and western critics . After release ,
it received downloadable content , along with an expanded edition in November of that year . It was also adapte
d into manga and an original video animation series . Due to low sales of Valkyria Chronicles II , Valkyria Chr
onicles III was not localized , but a fan translation compatible with the game 's expanded edition was released
in 2014 . Media.Vision would return to the franchise with the development of Valkyria : Azure Revolution for th
e PlayStation 4 . "
b' = = Gameplay = = '
```

Vectorize Input

In [26]:
```
def custom_standardization(input_data):
    lowercase = tf.strings.lower(input_data)
    text = tf.strings.regex_replace(lowercase, '[%s]' % re.escape(string.punctuation), '')
    return tf.strings.regex_replace(text, 'unk', "[UNK]")

vocab_size = 10000
sequence_length = 10
vectorize_layer = TextVectorization(standardize = custom_standardization,
                                    max_tokens = vocab_size,
                                    output_mode = 'int',
                                    output_sequence_length = sequence_length)

vectorize_layer.adapt(text_ds.batch(1024))
inverse_vocab = vectorize_layer.get_vocabulary()
print(len(inverse_vocab))
print(inverse_vocab[:20])
```

```
10000
['', '[UNK]', 'the', 'of', 'and', 'in', 'to', 'a', 'was', 'on', 'as', 's', 'that', 'for', 'with', 'by', 'is',
'at', 'from', 'it']
```

Batch and Prefetch

In [27]:
```
text_vector_ds = text_ds.batch(1024).prefetch(tf.data.AUTOTUNE).map(vectorize_layer).unbatch()

sequences = list(text_vector_ds.as_numpy_iterator())
print(len(sequences))

for seq in sequences[:5]:
    print(f'{seq} => {[inverse_vocab[i] for i in seq]}')
```

```
29119
[4795 4207  969    0    0    0    0    0    0    0] => ['valkyria', 'chronicles', 'iii', '', '', '', '', '',
'', '']
[   1   76 4795   81    1 4207  430    1    0    0] => ['[UNK]', 'no', 'valkyria', '3', '[UNK]', 'chronicles',
'japanese', '[UNK]', '', '']
[   2   64  127  365    5  283 2738   58    7  169] => ['the', 'game', 'began', 'development', 'in', '2010', 'c
arrying', 'over', 'a', 'large']
[  19  788   14  927 1614    5  739    4    8  731] => ['it', 'met', 'with', 'positive', 'sales', 'in', 'japa
n', 'and', 'was', 'praised']
[2199    0    0    0    0    0    0    0    0    0] => ['gameplay', '', '', '', '', '', '', '', '', '']
```

In [28]:
```
vocab = {}
for index, token in enumerate(inverse_vocab):
    vocab[token] = index

print(len(vocab))
```

```
10000
```

In [29]:
```
sentence = 'I am valkyria and am working as a machine learning with'
tokens = list(sentence.lower().split())
example_sequence = [vocab[word] for word in tokens]
print(example_sequence)
```

```
[59, 1610, 4795, 4, 1610, 652, 10, 7, 1021, 2952, 14]
```

In [30]:
```
window_size = 2
positive_skip_grams = tf.keras.preprocessing.sequence.skipgrams(example_sequence,
                                                                 vocabulary_size = vocab_size,
                                                                 window_size = window_size,
                                                                 negative_samples=0)

print(len(positive_skip_grams))
for target, context in positive_skip_grams[0]:
    print(f'({target}, {context}): ({inverse_vocab[target]}, {inverse_vocab[context]})')
```

```
2
(10, 1610): (as, am)
(652, 10): (working, as)
(10, 652): (as, working)
(10, 1021): (as, machine)
(652, 7): (working, a)
(7, 1021): (a, machine)
(10, 7): (as, a)
(4, 4795): (and, valkyria)
(14, 1021): (with, machine)
(14, 2952): (with, learning)
(1610, 4): (am, and)
(2952, 7): (learning, a)
(1021, 10): (machine, as)
(1610, 59): (am, i)
(7, 2952): (a, learning)
(4795, 1610): (valkyria, am)
(1610, 4795): (am, valkyria)
(4795, 59): (valkyria, i)
(1610, 4): (am, and)
(2952, 14): (learning, with)
(1610, 652): (am, working)
(1021, 2952): (machine, learning)
(4, 652): (and, working)
(4, 1610): (and, am)
(1610, 10): (am, as)
(1021, 14): (machine, with)
(2952, 1021): (learning, machine)
(7, 10): (a, as)
(652, 4): (working, and)
(1610, 4795): (am, valkyria)
(4, 1610): (and, am)
(7, 652): (a, working)
(59, 1610): (i, am)
(59, 4795): (i, valkyria)
(4795, 1610): (valkyria, am)
(4795, 4): (valkyria, and)
(1021, 7): (machine, a)
(652, 1610): (working, am)
```

Generate Training Data

In [89]:
```
def generate_training_data(sequences, window_size, num_ns, vocab_size, seed):
    targets, contexts, labels = [], [], []
    sampling_table = tf.keras.preprocessing.sequence.make_sampling_table(vocab_size)
    for sequence in tqdm(sequences):
        positive_skip_grams, _ = tf.keras.preprocessing.sequence.skipgrams(sequence,
                                                                            vocabulary_size= vocab_size,
                                                                            sampling_table = sampling_table,
                                                                            window_size = window_size,
                                                                            negative_samples = 0)
        if len(positive_skip_grams) != 0:
            for target_word, context_word in positive_skip_grams:
                context_class = tf.expand_dims(tf.constant([context_word], dtype = 'int64'), 1)
                negative_sampling_candidates, _, _ = tf.random.log_uniform_candidate_sampler(true_classes = con
                                                                                              num_true = 1,
                                                                                              num_sampled = num_
                                                                                              unique = True,
                                                                                              range_max = vocab_
                                                                                              seed = seed,
                                                                                              name = 'negative_s
                negative_sampling_candidates = tf.expand_dims(negative_sampling_candidates,
                                                              1)
                context = tf.concat([context_class, negative_sampling_candidates], 0)
                label = tf.constant([1] + list(np.repeat(0, num_ns)), dtype = 'int64')
                targets.append(target_word)
                contexts.append(context)
                labels.append(label)
    return targets, contexts, labels
```

In [90]:
```
SEED = 123
targets, contexts, labels = generate_training_data(sequences = sequences,
                                                   window_size = 2,
                                                   num_ns = 4,
                                                   vocab_size = vocab_size,
                                                   seed = SEED)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 29119/29119 [02:15<00:00, 21
4.35it/s]
```

In [91]:
```
print(len(targets), len(contexts), len(labels))
```

```
138341 138341 138341
```

In [92]:
```
print("target | context | label")
for target, context, label in zip(targets[:5], contexts[:5], labels[:5]):
    print(f"{target} | {context} | {label}")
    print("-"*80)
```

```
target | context | label
969 | [[4795]
 [ 236]
 [1030]
 [7758]
 [ 103]] | [1 0 0 0 0]
--------------------------------------------------------------------------------
4207 | [[4795]
 [  10]
 [  20]
 [   8]
 [ 424]] | [1 0 0 0 0]
--------------------------------------------------------------------------------
969 | [[4207]
 [2524]
 [   3]
 [   0]
 [3975]] | [1 0 0 0 0]
--------------------------------------------------------------------------------
4795 | [[4207]
 [  57]
 [9344]
 [ 192]
 [ 790]] | [1 0 0 0 0]
--------------------------------------------------------------------------------
4207 | [[969]
 [ 13]
 [  0]
 [ 64]
 [402]] | [1 0 0 0 0]
--------------------------------------------------------------------------------
```

Word2Vec

In [93]:
```
class Word2Vec(Model):
    def __init__(self, vocab_size, embedding_dim, num_ns, vocabulary):
        super(Word2Vec, self).__init__()
        self.vocabulary = vocabulary
        self.target_embedding = Embedding(vocab_size,
                                          embedding_dim,
                                          input_length = 1,
                                          name = "w2v_embedding")
        self.context_embedding = Embedding(vocab_size,
                                           embedding_dim,
                                           input_length = num_ns + 1)
        self.dots = Dot(axes = (3,2))
        self.flatten = Flatten()
    def call(self, pair):
        target, context = pair
        we = self.target_embedding(target)
        ce = self.context_embedding(context)
        dots = self.dots([ce, we])
        return self.flatten(dots)
```

In [94]:
```
num_ns = 4
embedding_dim = 256
word2vec = Word2Vec(vocab_size, embedding_dim, num_ns, vectorize_layer.get_vocabulary())
word2vec.compile(optimizer = 'adam', loss = tf.keras.losses.CategoricalCrossentropy(from_logits = True))
```

In [95]:
```
BATCH_SIZE = 1024
BUFFER_SIZE = 10000
dataset = tf.data.Dataset.from_tensor_slices(((targets, contexts), labels))
dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE, drop_remainder = True)
print(dataset)
```

```
<BatchDataset shapes: (((1024,), (1024, 5, 1)), (1024, 5)), types: ((tf.int32, tf.int64), tf.int64)>
```

In [96]:
```
dataset = dataset.cache().prefetch(tf.data.AUTOTUNE)
print(dataset)
```

```
<PrefetchDataset shapes: (((1024,), (1024, 5, 1)), (1024, 5)), types: ((tf.int32, tf.int64), tf.int64)>
```

In [97]:
```
history = word2vec.fit(dataset, epochs = 35)
```

```
Epoch 1/35
135/135 [==============================] - 7s 35ms/step - loss: 1.5986
Epoch 2/35
135/135 [==============================] - 5s 35ms/step - loss: 1.4680
Epoch 3/35
135/135 [==============================] - 5s 34ms/step - loss: 1.2992
Epoch 4/35
135/135 [==============================] - 5s 34ms/step - loss: 1.1334
Epoch 5/35
135/135 [==============================] - 5s 35ms/step - loss: 0.9750
Epoch 6/35
135/135 [==============================] - 5s 34ms/step - loss: 0.8327
Epoch 7/35
135/135 [==============================] - 5s 34ms/step - loss: 0.7085
Epoch 8/35
135/135 [==============================] - 5s 36ms/step - loss: 0.6023
Epoch 9/35
135/135 [==============================] - 5s 34ms/step - loss: 0.5130
Epoch 10/35
135/135 [==============================] - 5s 34ms/step - loss: 0.4390
Epoch 11/35
135/135 [==============================] - 5s 34ms/step - loss: 0.3782
Epoch 12/35
135/135 [==============================] - 5s 35ms/step - loss: 0.3285
Epoch 13/35
135/135 [==============================] - 5s 36ms/step - loss: 0.2880
Epoch 14/35
135/135 [==============================] - 5s 36ms/step - loss: 0.2549
Epoch 15/35
135/135 [==============================] - 5s 34ms/step - loss: 0.2278: 0s - loss:  - ETA: 0s - loss
Epoch 16/35
135/135 [==============================] - 5s 35ms/step - loss: 0.2054
Epoch 17/35
135/135 [==============================] - 5s 34ms/step - loss: 0.1868
Epoch 18/35
135/135 [==============================] - 5s 34ms/step - loss: 0.1712
Epoch 19/35
135/135 [==============================] - 5s 35ms/step - loss: 0.1582
Epoch 20/35
135/135 [==============================] - 5s 34ms/step - loss: 0.1472
Epoch 21/35
135/135 [==============================] - 5s 34ms/step - loss: 0.1378
Epoch 22/35
135/135 [==============================] - 5s 34ms/step - loss: 0.1298
Epoch 23/35
135/135 [==============================] - 5s 35ms/step - loss: 0.1229
Epoch 24/35
135/135 [==============================] - 5s 34ms/step - loss: 0.1170
Epoch 25/35
135/135 [==============================] - 5s 36ms/step - loss: 0.1118
Epoch 26/35
135/135 [==============================] - 5s 37ms/step - loss: 0.1073
Epoch 27/35
135/135 [==============================] - 5s 35ms/step - loss: 0.1033
Epoch 28/35
135/135 [==============================] - 5s 35ms/step - loss: 0.0998
Epoch 29/35
135/135 [==============================] - 5s 35ms/step - loss: 0.0967
Epoch 30/35
135/135 [==============================] - 5s 34ms/step - loss: 0.0939
Epoch 31/35
135/135 [==============================] - 5s 35ms/step - loss: 0.0915
Epoch 32/35
135/135 [==============================] - 5s 35ms/step - loss: 0.0893
Epoch 33/35
135/135 [==============================] - 5s 34ms/step - loss: 0.0873
Epoch 34/35
135/135 [==============================] - 5s 34ms/step - loss: 0.0855
Epoch 35/35
135/135 [==============================] - 5s 34ms/step - loss: 0.0839
```

In [98]:
```
word_vec = word2vec.get_layer('w2v_embedding').get_weights()[0]
word_vec = np.asarray(word_vec, dtype = 'float32')
```

In [99]:
```
vocab = vectorize_layer.get_vocabulary()
```

In [100…
```
out_v = io.open('./Results/vectors.tsv', 'w', encoding = 'UTF-8')
out_m = io.open('./Results/metadata.tsv', 'w', encoding = 'UTF-8')

for index, word in enumerate(vocab):
    if index == 0:
        continue
    vec = word_vec[index]
    out_v.write('\t'.join([str(x) for x in vec]))
    out_m.write(str(index) + "\t" + word + '\n')
out_v.close()
out_m.close()
```

Now go to Embedding Projector

In [24]:
```
# log_dir = './Results'
# weights = tf.Variable(word2vec.get_layer('w2v_embedding').get_weights()[0][1:])
# checkpoint = tf.train.Checkpoint(embedding = weights)
# checkpoint.save(os.path.join(log_dir, "embedding.ckpt"))

# config = projector.ProjectorConfig()
# embedding = config.embeddings.add()
# embedding.tensor_name = "embedding/.ATTRIBUTES/VARIABLE_VALUE"
# embedding.metadata_path = './Results/metadata.tsv'
# projector.visualize_embeddings(log_dir, config)
```

Visualize Word Embeddings

In [103…
```
from sklearn.manifold import TSNE
word_vectors = []
labels = []
for index, word in enumerate(vocab[:200]):
    if index == 0:
        continue
    vec = word_vec[index]
    label = word
    word_vectors.append(vec)
    labels.append(label)
    tsne_model = TSNE(perplexity = 40,
                      n_components = 2,
                      init = 'pca',
                      n_iter = 2500,
                      random_state = 123)
new_values = tsne_model.fit_transform(word_vectors)
x = []
y = []
for value in new_values:
    x.append(value[0])
    y.append(value[1])

plt.figure(figsize = (16,16))
for i in tqdm(range(len(x))):
    plt.scatter(x[i], y[i])
    plt.annotate(labels[i],
                 xy = (x[i], y[i]),
                 textcoords = 'offset points',
                 ha = 'right',
                 va = 'bottom')
plt.show()
```

```
  0%|                                                                          | 0/199 [00:00
<?, ?it/s]<ipython-input-103-0abe785b0b06>:26: UserWarning: You have used the `textcoords` kwarg, but not the `
xytext` kwarg.  This can lead to surprising results.
  plt.annotate(labels[i],
100%|██████████████████████████████████████████████████████████████████████████| 199/199 [00:00<00:00, 26
8.54it/s]
```